Corrigé TP0

1. Par tests ou par lecture du code, on constate que la fonction triangle prend en entrée trois entiers i, j, k et renvoie "cas 3" si les trois sont égaux, "cas 2" si parmi eux il y en a deux d'égaux, et cas 1 le reste de temps. Ces tests ne sont réalisés que si une première condition est vérifiée ; si elle ne l'est pas, triangle affiche "cas 0".

   Le nom de la fonction peut nous mettre sur la piste de son fonctionnement : interprétons i, j, k comme les longeurs des trois côtés d'un triangle. Alors :

   a) Le premier test vise à s'assurer que le triplet (i, j, k) correspond effectivement à des longueurs licites pour les côtés d'un triangle.

   b) On peut renommer les différents cas selon le type de triangle ainsi :

```
void triangle(int i, int j, int k)
{
    int eg = 0;
    if ((i+j <= k) || (i+k <= j) || (j+k <= i))
    {
        printf("ceci n'est pas un triangle");
    }
    else
    {
        if (i == j) {eg = eg + 1;}
        if (i == k) {eg = eg + 1;}
        if (j == k) {eg = eg + 1;}
        if (eg == 0) {printf("triangle scalène");}
        else
        {
            if (eg == 1) {printf("triangle isocèle");}
            else {printf("triangle équilatéral");}
        }
    }
}
```

2. La fonction divergence prend en entrée un flottant x et renvoie le plus petit entier naturel n tel que $H_n = \sum_{i=1}^{n} \frac{1}{i} \geq x$. L'existence d'un tel entier n, donc la terminaison de la fonction, est garantie par le fait que la série harmonique diverge. Elouan fait remarquer à juste titre que la représentation d'un réel en machine ne se fait pas de manière exacte, en conséquence de quoi, les réels $1/k$ pour k grand risquent fort d'être suffisamment proches de 0 pour être considérés comme égaux à zéro par un ordinateur. Autrement dit, la série harmonique converge lorsqu'on manipule des flottants et non des réels et la terminaison n'est plus assurée !

   Ce point illustre un comportement que nous avons souvent rencontré l'année dernière : lorsqu'on étudie la terminaison ou la correction d'une fonction, on considère généralement une fonction "idéale" manipulant des objets mathématiques et non des objets informatiques. Même une simple fonction d'ajout sur les entiers peut avoir des comportements aberrants mathématiquement : il suffit d'obtenir un dépassement de capacité !

3. La fonction a000045 prend en entrée un entier naturel n et renvoie le n-ème terme de la suite de Fibonacci F_n (qui porte le numéro A000045 selon la nomenclature de l'OEIS, Online Encyclopedia of Integer Sequences).

a) Le calcul du F_n de manière récursive induit un recouvrement important des appels récursifs ce qui aboutit à un algorithme de complexité exponentielle en temps et à des difficultés mémoire dues au dépassement rapide de la pile d'appels. La version impérative proposée contourne ces difficultés et conduit à un algorithme de complexité temporelle en $O(n)$ et de complexité spatiale en $O(1)$. Il faut savoir exposer ces arguments.

b) On peut par exemple renommer les variables de la façon suivante, afin de laisser transparaître la relation de récurrence définissant la suite de Fibonacci :

```
int fibonacci(int n)
{
    int f_nmoins1 = 0;
    int f_n = 1;
    for (int i = 2; i <= n; i++)
    {
        int f_nplus1 = f_n + f_nmoins1;
        f_nmoins1 = f_n;
        f_n = f_nplus1;
    }
    if (n <= 1) return n;
    else return f_n;
}
```

4. La fonction chaines invite l'utilisateur à entrer deux chaînes de caractères et affiche un message différent selon qu'elles sont égales, ont même longueur sans être égales ou n'ont pas même longueur.

a) On rappelle qu'une chaînes de caractères n'est rien d'autre qu'un tableau de caractères se terminant par la sentinelle '\0'. Pour calculer la longueur d'une chaîne, il suffit donc de la parcourir jusqu'à détecter ce caractère :

```
int longueur(char chaine[])
{
    int l = 0;
    while (chaine[l] != '\0')
    {
        l = l+1;
    }
    return l;
}
```

b) Une possibilité de modification des affichages est :

- A la place de "cas 0", afficher "les deux chaînes entrées sont égales".
- A la place de "cas 1", afficher "les deux chaînes ont même longueur sans être égales".
- A la place de "cas 2", afficher "les deux chaînes ne sont pas de même longueur".

Remarque : Le comportement de scanf fait que la spécification proposée pour la fonction chaines ne correspond pas tout à fait. Par exemple, si les chaînes entrées par l'utilisateur comportent un espace, on n'obtiendra pas le comportement attendu. Il serait sage de remplacer scanf par gets !

5. La fonction soleil modifie le triplet d'entiers à l'adresse qu'elle prend en entrée de sorte à ce que ces trois entiers soient triés dans l'ordre croissant. On peut la renommer trie_triplet par exemple.

a) Comme vu dans le cours sur les représentations d'entiers, les types non signés sur n bits, permettent de décrire l'intervalle $[\![0, 2^n - 1]\!]$. Pour les types signés sur n bits, on représente l'intervalle $[\![-2^{n-1}, 2^{n-1} - 1]\!]$. On obtient donc :

Type	Intervalle représenté
uint8_t	$[\![0, 255]\!]$
int8_t	$[\![-128, 127]\!]$
uint32_t	$[\![0, 2^{32} - 1]\!]$
int32_t	$[\![-2^{31}, 2^{31} - 1]\!]$
uint64_t	$[\![0, 2^{64} - 1]\!]$
int64_t	$[\![-2^{63}, 2^{63} - 1]\!]$

b) On note (a_0, a_1, a_2) le triplet intial. La première conditionnelle stocke le plus petit entier parmi $\{a_0, a_1\}$ dans la variable u et le plus grand dans v. Pour réordonner le triplet, il ne reste plus qu'à déterminer la position de c par rapport à l'intervalle $[u, v]$: soit avant (premier cas), soit dedans (deuxième cas), soit après (troisième cas).

6. On constate tout d'abord que la variable BASE contient $32 = (100000)_2$ et la variable MASK, $31 = (11111)_2$. Les réponses aux questions ci-après et le nom de la fonction, permettent de comprendre que extraction(n, k) renvoie la valeur du nombre a_k devant 32^k dans l'écriture de n en base 32. En effet, l'opération $n \gg k \times$BLOCK_SIZE permet de calculer (la partie entière de) $n/32^k$: les BLOCK_SIZE $= 5$ derniers bits de cet entier forment justement l'écriture binaire de a_k. Comme MASK a pour écriture binaire 5 uns précédés de zéros, le "et" bitwise permet de les extraire.

   On observe sur cet exemple les limites de la stratégie consistant à faire quelques tests au hasard pour inférer le comportement de la fonction. Dans ces conditions, il est bien préférable de faire une batterie de tests ; par exemple, de calculer extraction(n,k) pour tout $n \in [\![0, 100]\!]$ avec k fixé (par exemple à 0 ou 1 pour commencer) à l'aide d'une boucle pour.

   a) Dans chacun des cas, on écrit les arguments de l'opérateur bitwise considéré sous forme binaire. On a $10 = (1010)_2$ et $9 = (1001)_2$. Ainsi, l'écriture binaire de 10&9 est 1000, ce qui correspond à 8 ; celle de 10|9 est 1011 ce qui correspond à 11 et celle de 10^9 à 0011, c'est-à-dire 3.

      De même, l'écriture binaire de 3 est 11 ce qui décalé deux fois à gauche donne 1100, soit 12 et l'écriture binaire de 42 est 101010 ce qui décalé à droite deux fois donne 1010 : l'écriture en binaire de l'entier 10.

   b) L'opération $n \ll i$ décale la représentation binaire de n de i chiffres vers la gauche en rajoutant des 0 à droite. Sous réserve qu'il n'y a pas de dépassement de capacité, cela revient à multiplier n par 2^i. L'opération $n \gg i$ décale la représentation binaire de n de i bits vers la droite, ce qui revient à calculer la partie entière inférieure de $n/2^i$.

   c) La négation bit à bit de 10 dépend du type de 10 : on obtiendra un résultat différent selon que 10 est considéré sur 8 ou 16 bits par exemple.

7. La fonction kramp prend en entrée un entier naturel n et renvoie un tableau de taille n contenant $(i + 1)!$ à la case i. Ce tableau est alloué sur le tas, il faut donc prendre garde à libérer la mémoire qu'il utilise après un test afin de ne pas provoquer de fuite mémoire. On peut à cet effet construire une fonction de test dédiée qui affiche le contenu de kramp(n) puis effectue les libérations nécessaires :

```c
void test_facto(int n)
{
    int* t = kramp(n);
    for (int i = 0; i < n ; i++)
    {
        printf("%d ", t[i]);
    }
    free(t);
}
```

Christian Kramp fut l'un des premiers mathématiciens à utiliser la notation $i!$ pour désigner la factorielle de i.

8. La fonction elucide (anagramme d'Euclide) calcule pédestrement le pgcd des deux entiers qu'elle prend en entrée, à condition que ceux ci soient strictement positifs tous les deux. Remarquons que si l'un des deux entiers en entrée est nulle, la boucle tant que ne termine pas et que lorsque l'un des deux est négatif, ce n'est pas le pgcd des entrées qui est calculé : l'assertion en début de code est indispensable. La fonction etienne permet de calculer le pgcd de deux entiers sans restriction de signe (et plus encore !) avec une meilleure complexité qui plus est.

9. La fonction etienne modifie les entiers pointés par x, y et z de sorte à ce que

$$*x = \text{pgcd}(a,b) = a \times *y + b \times *z$$

Autrement dit, elle calcule le pgcd de a et b ainsi que les coefficients de Bézout (Etienne de son prénom) associés. On pourrait la renommer coefficients_Bezout par exemple.

a) Il est impossible en C d'avoir une fonction qui renvoie plusieurs valeurs. On peut contourner ce problème en créant un nouveau type qui encapsule plusieurs valeurs ou en passant une partie des arguments par référence : c'est cette option qui a été choisie ici.

b) La preuve complète est laissée en exercice. Elle repose sur le fait que lorsque $a, b \in \mathbb{Z}$ et que q, r sont les quotient et reste dans la division euclidienne de a par b (donc, $a = bq + r$) on a :

- $pgcd(a, b) = pgcd(b, r) = d$.
- Si u, v dont des entiers tels que $d = bu + vr$ alors $d = va + b(u - vq)$. Autrement dit, si (u, v) est un couple de Bézout pour (b, r) alors $(v, u - vq)$ en est un pour (a, b).

Remarquons que la conditionnelle finale dans la fonction etienne permet de s'assurer que le pgcd calculé est positif (s'il ne l'est pas, on multiplie la relation de Bézout trouvée par -1).